AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ

KATEDRA INFORMATYKI STOSOWANEJ

Praca dyplomowa inżynierska

Implementacja systemu uwierzytelniania z zastosowaniem Negatywnych Baz Danych

Implementation of authentication system using Negative Databases

Autor: *Grzegorz Nieużyła*

Kierunek studiów: *Informatyka*

Opiekun pracy: *dr inż. Piotr Szwed*

Kraków, 2020

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Spis treści

1. Wprowadzenie

1.1. Cele pracy

Celem niniejszej pracy jest implementacja i przetestowanie systemu uwierzytelniania oferującego większe bezpieczeństwo niż standardowy schemat generowania skrótu hasła za pomocą funkcji generacji klucza (np. *PBKDF2*, *bcrypt*) i przechowywaniu w standardowej (pozytywnej) bazie danych.

Założeniem systemu jest zamienienie reprezentacji w sposób jawny na Negatywną Bazę Danych (NDB) co pozwoli dodać dodatkową warstwę bezpieczeństwa która znacząco utrudni uzyskanie haseł użytkownika w przypadku wykradzenia bazy danych.

W tym celu opisałem różne algorytmy prezentowane w dostępnej literaturze, przedstawiłem schemat ich działania i porównałem je pod względem bezpieczeństwa.

Rezultatem wyjściowym algorytmów generacji NDB jest zbiór ciągów tekstowych, które można jednoznacznie sprowadzić do zbioru formuł logicznych CNF, dlatego przeprowadzone zostały testy z wykorzystaniem solwerów SAT mające na celu zasymulowanie ataku na powstałą NDB.

1.2. Zawartość pracy

2. Negatywne Bazy Danych - opis teoretyczny

2.1. Opis działania

Główną operacja wykonywalną na NDB jest sprawdzenie czy dany rekord znajduje się w bazie. Przyjmując U jako oznaczenie uniwersum języka binarnego o długości l a DB jako zbiór wszystkich rekordów, każdy o długości l, NDB przechowuje zbiór $U - DB$ [1]. Takie dane są niepraktycznie do zareprezentowania w postaci nieskompresowanej z uwagi na wielkość, dlatego stosuje się wyrazy nad alfabetem $\{0, 1, *\}$ gdzie symbol $*$ może oznaczać zarówno 0 lub 1 w jawnej reprezentacji bitowej. Pozycje na których znajduje się wartość 0 lub 1 są *ustalone* a z wartością $*$ - *nieustalone*.

Każdy taki wyraz odpowiada jednemu lub wielu elementom $U - DB$ i jest sprowadzany do formuły logicznej (Tabela 2.1). Z założenia algorytm sprawdzający przynależność do DB sprawdza czy jakakolwiek formuła z NDB jest spełniana przez dany rekord. Dane znajdują się w DB wtedy i tylko wtedy gdy żadna formuła nie zostanie spełniona.

Taki model działania wymusza na danych stałą wielkość, co jednak nie stanowi problemu w przypadku przechowywania skrótów haseł które mają stałą, zależną od konkretnego algorytmu długość. Dla danych o zmiennych rozmiarach (np. nazwy użytkownika) można zastosować funkcję hashującą lub algorytmy zwiększające długość ciągu bitowego do stałej wartości typu PKCS#5 lub PKCS#7. Należy jednak pamiętać, że zwiększenie długości rekordu znacznie wydłuża czas generacji bazy oraz zajmowaną pamięć dla niektórych algorytmów.

rekord NDB	formuła logiczna
011*	$\neg x_1 \wedge x_2 \wedge x_3$
001*	$\neg x_1 \wedge \neg x_2 \wedge x_3$
1*1*	$x_1 \wedge x_3$
0*0*	$\neg x_1 \wedge \neg x_3$
1*00	$x_1 \wedge \neg x_3 \wedge \neg x_4$

Tabela 2.1. Reprezentacja formuł logicznych za pomocą NDB kodującej zbiór $DB = \{1001,\ 1101,\ 1110\}$

2.2. Zastosowanie w systemach uwierzytelniania

NDB może być wykorzystana w każdym systemie, gdzie podstawową operacją na danych jest sprawdzenie czy dany rekord znajduje się w bazie. Jednym z najpopularniejszych systemów uwierzytelniania jest metoda oparta na loginie i haśle. Użytkownik danej aplikacji przy zakładaniu konta podaje hasło, które następnie warstwa serwerowa danej aplikacji przechowuje jako wynik nieodwracalnej funkcji hashującej.

W przypadku nieautoryzowanego dostępu do bazy danych i używanego algorytmu uzyskiwania skrótu z hasła, atakujący może uzyskać wartość pierwotną mało skomplikowanych haseł za pomocą np. metody słownikowej. Modyfikując powyższy algorytm składując skróty jako rekordy w NDB uniemożliwiamy łatwą iterację wszystkich danych, jednocześnie pozostawiając łatwy dostęp do informacji czy użytkownik o podanym loginie i haśle ma dostęp do aplikacji.

2.3. Algorytmy generacji Negatywnych Baz Danych

2.3.1. Algorytm prefiksowy

Najprostszym ze sposobów generowania Negatywnych Baz Danych jest zaproponowany przez Fernando Esponda algorytm prefiksowy [1, 2]. Został on opracowany w celu udowodnienia że proces generowania NDB z rekordów DB jest możliwy w rozsądnej złożoności czasowej i pamięciowej.

Algorytm 1: Algorytm prefiksowy

Dane: DB - zbiór rekordów do zareprezentowania w NDB, l - długość rekordu DB

Wynik: Zbiór rekordów NDB

$Prefix_n(V)$ - Prefiks n-znakowy rekordu V
$len(V)$ - Długość rekordu V
W_i = { };
i = 0;
while $i < l$ **do**
    W_{i+1} = Zbiór wszystkich $i + 1$-znakowych ciągów bitowych V_p nie będących prefiksem żadnego rekordu DB i dla których $Prefix_i(V_p) \in W_i$
    foreach V_p *in* W_{i+1} **do**
        Stwórz rekord NDB o długości l którego V_p jest prefiksem a na pozostałych pozycjach jest symbol $*$ i dodaj do zbioru wyjściowego NDB
    end
    i = i + 1;
    W_i = Zbiór wszystkich i-znakowych prefiksów rekordów DB
end

Powyższa metoda polega na generowaniu coraz dłuższych prefiksów które nie pokrywają się ze zbiorem DB. W ten sposób na początku tworzone są rekordy odpowiadające znacznej części $U - DB$. Czasami występuje potrzeba zdefiniowania pewnych rekordów explicite bez wykorzystania symbolu $*$ jeżeli każdy możliwy prefiks jest także prefiksem rekordu z DB. Przykładowy wynik działania znajduje się w tabeli 2.2.

DB	U - DB	NDB
0000	0001	10**
0110	0011	010*
0010	0100	111*
1101	0101	0001
	0111	0011
	1000	0111
	1001	1100
	1010	
	1011	
	1100	
	1110	
	1111	

Tabela 2.2. Rezultat działania algorytmu prefiksowego

Algorytm prefiksowy jest deterministyczny i każdy powstały rekord reprezentuje unikalną, nie pokrywającą się część $U - DB$ [1]. Powoduje to, że algorytm uzyskiwania zbioru DB z otrzymanej NDB

nie wymaga sprowadzenia do problemu SAT. Wystarczy jedynie odpowiednio posortować rekordy i wyznaczyć przedziały pomiędzy nimi, co pokazuję w algorytmie 2.

Algorytm 2: Algorytm odwracający NDB wygenerowaną za pomocą algorytmu prefiksowego

Dane: NDB - zbiór rekordów do odwrócenia, l - długość rekordu NDB

Wynik: Zbiór rekordów DB

Posortuj zbiór NDB

V_l = rekord DB wypełniony symbolami '0' o długości l

foreach V ***in*** NDB **do**

  V_h = V z zamienionymi symbolami '$*$' na '0'

  R = $Rekordy\_DB\_W\_Przedziale(V_l, V_h)$

  Dodaj R do zbioru wynikowego

  $V_l = V$ z zamienionymi symbolami '$*$' na '1'

end

V_h = rekord DB wypełniony symbolami '1' o długości l

R = $Rekordy\_DB\_W\_Przedziale(V_l, V_h)$

Dodaj R do zbioru wynikowego

Algorytm 3: Rekordy_DB_W_Przedziale

Dane: V_1, V_2 - rekordy DB

Wynik: Zbiór rekordów DB w przedziale (V_1, V_2)

Dodaj binarnie 1 do V_1

while $V_1 \neq V_2$ **do**

  Dodaj V_1 do zbioru wynikowego.

  Dodaj binarnie 1 do V_1

end

Czas wykonywania procedury wynosi $O(l|DB|)$, jednak w przypadku zapisywania wyniku do bazy wzrasta do $O(l^2|DB|)$ gdyż konieczna jest serializacja każdego rekordu. Złożoność pamięciowa dla optymalnej implementacji wynosi $O(l|DB|)$ w przypadku gdy poprzednio generowane rekordy NDB nie są przetrzymywane w pamięci. Dla danego zbioru DB generowane jest $O(l|DB|)$ rekordów co sprowadza się do wielkości powstałej bazy danych wynoszącej $O(l^2|DB|)$.

Przestawiony powyżej algorytm odwracający (2) wykonuje się w czasie $O(|NDB|) = O(l|DB|)$ co jest wynikiem lepszym od procedury generacyjnej z zapisem do bazy.

2.3.2. Algorytm *Randomize_NDB*

Algorytm prefiksowy generuje poprawne rekordy NDB, jednak nie jest praktyczny w żadnych zastosowaniach związanych z bezpieczeństwem, ponieważ wynik jego działania jest stosunkowo prosto sprowadzić do postaci pozytywnej. Aby temu zaradzić, Fernando Esponda w swojej pracy[1] zaproponował niedeterministyczny algorytm mający na celu wprowadzić rekordy które nie odpowiadają jedynie prefiksom elementów z $U - DB$ i są trudniejsze do odwrócenia.

Algorytm 4: Algorytm *Randomize_NDB*

Dane: DB - zbiór rekordów do zareprezentowania w NDB, l - długość rekordu DB

Wynik: Zbiór rekordów NDB

$Prefix_n(V)$ - Prefiks n-znakowy rekordu V
$len(V)$ - Długość rekordu V
π = losowa permutacja o długości $|V_{pe}|$
W_i = zbiór wszystkich ciągów l-bitowych
$\pi(DB) \equiv \{\pi(V) \mid V \in DB\}$
$i = \lceil log_2(l) \rceil$;
while $i < l$ ***and*** $W_i \neq \emptyset$ **do**
    W_{i+1} = Zbiór wszystkich $i+1$-znakowych ciągów bitowych V_p nie będących prefiksem żadnego rekordu $\pi(DB)$ i dla których $Prefix_i(V_p) \in W_i$
    foreach V_p ***in*** W_{i+1} **do**
        Dopełnij V_p do długości l wstawiając znaki nieustalone '$*$' na końcu
        j = losowa liczba z przedziału $[1, l]$
        for $k = 1$ ***to*** j **do**
            n = losowa liczba z przedziału $[1, log_2(l)]$
            P = n losowych nieustalonych pozycji z V_p
            X = Zbiór rekordów powstałych przez zastąpienie pozycji $\in P$ przez wszystkie możliwe kombinacje bitowe (2^n rekordów)
            foreach V_q ***in*** X **do**
                V_{pg} = Pattern_Generate($\pi(DB)$, V_q)
                Dodaj $\pi^{-1}(V_{pg})$ do zbioru rekordów wyjsciowych
            end
        end
    end
    $i = i + 1$
    W_i = Zbiór wszystkich i-znakowych prefiksów rekordów DB
end

Algorytm ten działa na podobnej zasadzie co algorytm prefiksowy (rozdział 2.3.1) z pewnymi modyfikacjami. Na początku kolejność bitów w DB jest mieszana za pomocą losowej permutacji aby pozycje zdefiniowane w generowanej prefiksowej NDB nie były skumulowane na początku wyrazów. Następnie dla każdego powstałego negatywnego rekordu losuje się n pozycji nieustalonych i zastępuje się go równoznacznym zbiorem rekordów które mają te pozycje ustalone. Powstałe ciągi są dodatkowo zaciemniane przez wstawienie na losowych pozycjach zamiast bitu zdefiniowanego znak $*$ zgodnie z algorytmem *Pattern_Generate*.

Wynik algorytmu jest niedeterministyczny co powoduje że dla takich samych zbiorów DB rezultat może się różnić. Generacja wielu redundantnych rekordów zwiększa odporność otrzymanej bazy na próby przywrócenia do postaci pozytywnej, jednak wiąże się to z ze zwiększeniem objętości NDB średnio $\frac{l^2}{2}$ razy w stosunku do algorytmu prefiksowego.

Algorytm 5: Algorytm Pattern_Generate

Dane: DB - zbiór rekordów do zareprezentowania w NDB, V_q - rekord NDB do zaciemnienia

Wynik: Zaciemniony rekord NDB

π = losowa permutacja o długości $|V_{pe}|$
SIV = { } // wektor zmienionych bitów
for $i = 1$ *to* $|V_q|$ **do**
    $\pi(V_q)'$ = $\pi(V_q)$ z elementem o indeksie i zamienionym na symbol $*$
    if *istnieje rekord w DB pokrywający się z $\pi(V_q)'$* **then**
        Dodaj i i bit o indeksie i do SIV
        $\pi(V_q) = \pi(V_q)'$
    end
end
$t =$ losowa liczba z przedziału $[0, |SIV|]$
if $t > |SIV|$ **then**
    $R = SIV$
else
    $R = t$ losowych bitów z SIV
end
V_k = $\pi(V_q)$
for $indeks$, bit *in* R **do**
    $V_k[indeks]$ = bit
end
Zwróć $\pi^{-1}(V_k)$

Zmiany względem algorytmu prefiksowego oraz procedura *Pattern_Generate* gwarantują że żaden ze zmodyfikowanych rekordów nie będzie odpowiadał żadnemu elementowi DB oraz że wynik działania pozostanie kompletnym odzwierciedleniem $U - DB$, ponieważ każda modyfikacja rekordu rozszerza zakres pokrywanych ciągów wprowadzając redundantne informacje. W [1] przedstawiony jest dowód że problem rekonstrukcji DB z NDB jest NP-trudny (każdą instancję 3-SAT można sprowadzić do NDB).

Powyższy algorytm w teorii jest zdolny do tworzenia trudnych instancji, ale nie posiada żadnych mechanizmów umożliwiających ingerencję w jego działanie – wynik jest zawsze losowy. Prawdopodobieństwo uzyskania trudnej do złamania kombinacji rekordów jest bardzo małe, co powoduje że w znacznej większości przypadków wynikowa NDB może być odwrócona niemal natychmiast przez współczesne solwery SAT, co pokazuję w dalszych rozdziałach.

DB	U - DB	NDB
0000	0001	*001
0110	0011	00*1
0010	0100	0011
1101	0101	010*
	0111	0*11
	1000	1**0
	1001	10**
	1010	101*
	1011	**11
	1100	1*1*
	1110	1100
	1111	111*

Tabela 2.3. Rezultat działania algorytmu *Randomize_NDB*

2.3.3. Algorytm *0-Hidden*

Następujące algorytmy powstały przez zastosowanie procedur generowania trudnych instancji SAT. Jedną z nich jest **0-Hidden** służąca do generacji formuł 3-SAT nie posiadających rozwiązania, co pozwala na testowanie solwerów pod względem wykrywania braku spełnialności [3].

Algorytm 6: Algorytm *0-Hidden*

Dane: l - liczba zmiennych, r - współczynnik ilości klauzul

Wynik: Zbiór klauzul 3CNF

```
n = l * r
W = { }
while |W| ≠ n do
    Wybierz 3 losowe zmienne
    Stwórz klauzulę 3CNF używając wylosowanych zmiennych, z losowymi znakami
    Dodaj klauzulę do zbioru wynikowego W
end
```

Powyższy algorytm generuje formułę CNF, która z dużym prawdopodobieństwem nie jest spełnialna i jest trudna do rozwiązania przez solwery SAT [3, 4]. Modyfikując parametr r możemy wpłynąć na rozmiar formuły, wartość $r \approx 4.27$ jest wartością graniczną powyżej której problem prawie na pewno nie ma rozwiązania [3].

Z perspektywy Negatywnych Baz Danych można przyjąć że powstała formuła odpowiada zbiorowi $U - DB$, jeśli DB jest zbiorem pustym.

2.3.4. Algorytmy *1-Hidden* i *2-Hidden*

Algorytmy ***1-Hidden*** i ***2-Hidden*** są skonstruowane podobnie jak *0-Hidden*, z tą różnicą, że formuła wyjściowa ma odpowiednio co najmniej (i w znacznej większości dokładnie) jedno lub dwa rozwiązania.

W celu testowania możliwości solwerów SAT co do znajdowania przypisania spełniającego daną formułę można generować losowe formuły (Algorytm 6) i odrzucać klauzule które przeczą ukrytemu rozwiązaniu A (*1-Hidden*). Jednak spowoduje to, że rozkład losowy zostanie zaburzony i solwery mogą to „poczuć”, co doprowadzi je do ukrytego rozwiązania [4].

Aby temu przeciwdziałać można jednocześnie ukryć przypisania A oraz $\neg A$ (*2-Hidden*), co spowoduje że algorytm przeszukujący będzie równoważnie „przyciągany” przez dwa przeciwne rozwiązania.

Powyższy schemat działania jest wykorzystany w procedurze ***Q-Hidden*** będącej rozszerzeniem tego konceptu.

2.3.5. Algorytm *Q-Hidden*

Metoda ***Q-Hidden*** do generacji trudnych formuł K-SAT zaproponowana w [3] rozszerza możliwości algorytmów z rozdziałów 2.3.3 i 2.3.4 przez dodanie parametru prawdopodobieństwa $q \in (0, 1)$. Każda wygenerowana klauzula składa się z k zmiennych których przypisanie pokrywa się z $t > 0$ literałów z prawdopodobieństwem q^t (gdzie t to numer kolejnego zgodnego literału). Oznacza to że im mniejsza wartość q tym bardziej formuła będzie wskazywać w przeciwnym kierunku niż ukryte rozwiązanie A.

Na podstawie tego schematu powstał algorytm mający na celu rozwiązanie problemów z algorytmami prefiksowym i Randomize_NDB tj. generowanie zbiorów rekordów które łatwo odwrócić za pomocą solwerów SAT[5]

Algorytm 7: Algorytm *Q-Hidden*

Dane: s - ciąg bitowy, l - długość rekordu, r - wsp. ilości klauzul
k - ilość ustalonych bitów w klauzuli, q - wsp. prawdopodobieństwa
Wynik: Zbiór rekordów NDB

```
n = l * r
NDB = {}
while |NDB| ≠ n do
    Υ = k różnych losowych pozycji w przedziale [1, l]
    V = rekord NDB o długości l wypełniony znakami '*'
    foreach i in Υ do
        V[i] = s[i]
    end
    d = 0
    while d ≠ 1 do
        foreach i in Υ do
            p = losowa liczba rzeczywista z przedziału [0, 1]
            if q > p then
                V[i] = ¬V[i]
                d = 1
            end
        end
    end
    Dodaj rekord V do NDB
end
```

W przeciwieństwie do poprzednich metod, za pomocą algorytmu *Q-Hidden* przy zastosowaniu odpowiednich parametrów q, r i k jest możliwe praktycznie stworzenie instancji NDB rozwiązywalnych

NDB		
*10*1	11**0	**100
11**0	**100	*000*
**100	*000*	0*01*
000	0*01*	*0*01
0*01*	*0*01	01*1*
*0*01	01*1*	1*0*1
01*1*	1*0*1	1**01

Tabela 2.4. Rezultat działania algorytmu *Q-Hidden* dla ciągu bitowego 10010 przy parametrach $k = 3$, $q = 0.5$ i $r = 4.2$

przez solwery SAT w czasie potęgowym tj. przez całkowite przeszukanie, co powoduje że dla odpowiednio dużych l odzyskanie zabezpieczonego ciągu jest niepraktyczne.

Takie podejście do problemu ma szereg konsekwencji. Pierwszą z nich jest to że wielkość otrzymanej bazy nie zależy od przechowywanego rekordu i można ją kontrolować za pomocą parametru r - generowane jest dokładnie $l * r$ rekordów. Nie ma jednak możliwości zawarcia w jednej instancji NDB więcej niż jednego ciągu bitowego - można zakodować jedynie zero lub jeden rekord.

W [5] razem z przedstawieniem algorytmu 7 opisany jest przykładowe zastosowanie tego schematu w systemach rzeczywistych wymagających znacznie większych ilości danych. Polega na przechowywaniu zbioru osobnych NDB dla każdego pozytywnego rekordu. Metoda zapytania do tak skonstruowanej bazy polega na sprawdzeniu po kolei wszystkich pojedynczych NDB. Jeżeli rekord nie pokrywa się z żadną formułą w jakiejkolwiek pod-bazie to przyjmuje się że znajduje się w bazie głównej.

Negatywne Bazy Danych nie przechowujące żadnych danych okazują się przydatne w takim rozwiązaniu ponieważ „puste” bazy stworzone przez algorytm *0-Hidden* (zmodyfikowany w celu generowania formuł o wymaganej liczbie literałów w klauzuli) są nierozróżnialne od tych wygenerowanych przez algorytm *Q-Hidden*. Pozwala to na ukrycie liczby rekordów pozytywnych.

Istotnym problemem jest fakt, że nie ma gwarancji, że rezultat działania tego algorytmu będzie pokrywał cały zbiór $U - \{s\}$, nawet powyżej wartości granicznej $r \approx 4.27$. Jedną z opcji jest dalsze zwiększenie parametru r żeby jeszcze bardziej zmniejszyć prawdopodobieństwo zawarcia nadmiarowych ciągów, ale wprowadza to dodatkowe, redundantne informacje co powoduje że czas potrzebny na odwrócenie NDB maleje. Dlatego wskazane jest ustawienie niskiej wartości r i dodanie do każdego rekordu dodatkowej informacji wskazującej na jego poprawność np. bitu parzystości, kodu CRC lub wyniku funkcji skrótu. W takiej sytuacji prawdopodobieństwo wystąpienia niechcianych danych maleje 2^c-krotnie, gdzie c oznacza liczbę dodatkowych bitów.

Rozkład wartości nadmiarowych ciągów bitowych nie jest jednostajny - dodatkowe wystąpienia będą bliskie s w odległości Hamminga (tj. poszczególne bity będą się różniły w niewielkiej ilości), dlatego metoda CRC maksymalizująca odległość Hamminga dla podobnych napisów może być szczególnie efektywna. [5]

Porównanie poszczególnych metod przedstawiam w rozdziale **??**.

2.3.6. Algorytm *K-Hidden*

K-Hidden jest modyfikacją algorytmu *Q-Hidden*. Zamiast pojedynczego parametru prawdopodobieństwa q stosuje się w nim wektor parametrów $\{p_1 \dots p_k\}$ w celu zwiększenia ilości stopni swobody algorytmu [6]. W ten sposób możemy dowolnie kontrolować dystrybucję różnych typów klauzul - w zależności od liczby pokrywających się bitów z ciągiem wejściowym.

Algorytm 8: Algorytm *K-Hidden*

Dane: s - ciąg bitowy, l - długość rekordu, r - wsp. ilości klauzul
k - ilość ustalonych bitów w klauzuli, $\{p_1 \dots p_k\}$ - wsp. prawdopodobieństwa
Wynik: Zbiór rekordów NDB

$n = l * r$
$NDB = \{\}$
Q = $\{Q_0, Q_1 \dots Q_k\} : Q_0 = 0, Q_i = p_1 + \cdots + p_i$
while $|NDB| \neq n$ **do**
    rnd = losowa liczba rzeczywista z przedziału $(0, 1)$
    $i = i : Q_{i-1} \leq rnd \leq Q_i$
    V = rekord z k ustalonymi bitami, pokrywający się z s na $k - i$ losowych pozycjach
    Dodaj rekord V do NDB
end

Przed generacją każdego kolejnego rekordu losuje się liczbę z przedziału $(0, 1)$ i na podstawie jej i wektora parametrów p_i wybiera się odpowiedni typ rekordu - mający dokładnie i pokrywających się zmiennych gdzie i jest indeksem najmniejszego parametru większego od wylosowanej liczby.

Rezultat działania ma podobne cechy co wygenerowany przez *Q-Hidden* - możliwość zakodowania tylko jednego napisu i brak gwarancji pokrycia całego zbioru $U - \{s\}$.

W literaturze pojawia się także zaproponowany wcześniej algorytm *p-Hidden* będący szczególnym przypadkiem *K-Hidden* dla $k = 3$ [7]. Analogicznie dla przypadku gdy wektor parametrów p jest następujący:

$$p_i = \frac{\binom{k}{i} q^i}{(1+q)^k - 1}, i = 1 \dots k - 1$$

algorytm jest równoważny *Q-Hidden*. [6]

2.3.7. Algorytm hybrydowy

Algorytm hybrydowy jest połączeniem algorytmu prefiksowego oraz *Q-Hidden* [8]. Ma na celu zapewnienie, że wygenerowana baza będzie zawierać całe uniwersum napisów bez jednego, ukrytego rekordu i jednocześnie będzie odporna na odwrócenie. Baza wynikowa jest równoważna formule 3-CNF.

Algorytm 9: Algorytm hybrydowy

Dane: s - ciąg bitowy, l - długość rekordu, r - wsp. ilości klauzul
q - wsp. prawdopodobieństwa

Wynik: Zbiór rekordów NDB

```
1 NDB = GenComplete(s, l)
2 NDB = MakeHardReverse(NDB, s, l, r, q)
```

Proces generacji składa się z dwóch etapów. Pierwszy, procedura *GenComplete*, tworzy kompletny zbiór $l + 4$ rekordów pokrywający całe $U - \{s\}$. Na początku tworzy $l - 2$ rekordy kolejno zaprzeczając każdemu bitowi oprócz dwóch ostatnich i losuje dwie dodatkowe pozycje przed nim ustawiając je zgodnie z s. Następnie generuje dwa rekordy dla różniącej się pozycji drugiej i cztery dla pierwszej.

NDB jest tworzona dla losowej permutacji napisu s a następnie każdy rekord jest przekształcany z powrotem przez permutację odwrotną w celu utrudnienia analizy przez pomieszanie kolejności bitów.

W kolejnym etapie tworzone są pozostałe $l * r - (l + 4)$ rekordy przez zastosowanie procedury *MakeHardReverse* (Algorytm 11) - równoznacznej *Q-Hidden* dla $k = 3$.

Wynik działania tej metody dla nieznanego napisu s jest nierozróżnialny od baz utworzonych przez algorytmy *0-Hidden*, *1-Hidden*, *2-Hidden*, *Q-Hidden* i *K-Hidden* przy $k = 3$ i identycznym r. Różnić się będą jedynie rozkładem pozycji ustalonych zgodnych i niezgodnych z ukrytym napisem.

11000
*1*01
1*01*
111**
100**
101**
0**00
0**10
0**01
0**11

Tabela 2.5. Rezultat działania procedury *GenComplete* dla $\pi(s) = 11000$ przed zastosowaniem permutacji odwrotnej na rekordach

Algorytm 10: GenComplete

Dane: s - ciąg bitowy, l - długość rekordu

Wynik: Zbiór rekordów NDB

π = losowa permutacja o długości l
for $k = l$ ***to*** 3 **do**
    V = rekord NDB o długości l zapełniony symbolami '$*$'
    $V[k]$ = $\neg\pi(s)[k]$
    i, j = dwie różne liczby z przedziału $[1, k]$
    $V[i]$ = $\pi(s)[i]$
    $V[j]$ = $\pi(s)[j]$
    Dodaj rekord V do NDB
end
V = rekord NDB o długości l zapełniony symbolami '$*$'
$V[1]$ = $\pi(s)[1]$
$V[2]$ = $\neg\pi(s)[2]$
i = losowa liczba z przedziału $[3, l]$
$V[i]$ = 0, Dodaj rekord V do NDB
$V[i]$ = 1, Dodaj rekord V do NDB
i = losowa liczba z przedziału $[2, l]$
for $b = 0$ *to* 1 **do**
    V = rekord NDB o długości l zapełniony symbolami '$*$'
    $V[1]$ = $\neg\pi(s)[1]$
    j = losowa liczba z przedziału $[2, l]$, różna od i
    $V[i]$ = b
    $V[j]$ = 0, Dodaj rekord V do NDB
    $V[j]$ = 1, Dodaj rekord V do NDB
end
Zwróć $\pi^{-1}(NDB)$

Algorytm 11: MakeHardReverse

Dane: s - ciąg bitowy, l - długość rekordu, r - wsp. ilości klauzul
q - wsp. prawdopodobieństwa

Wynik: Zbiór rekordów NDB

$n = l * r$
while $|NDB| \neq n$ **do**
    Υ = 3 różne losowe pozycje w przedziale $[1, l]$
    V = rekord NDB o długości l wypełniony znakami '$*$'
    X = losowe przypisanie $\in \{0, 1\}^3$
    $u = 3$
    foreach $i = 1$ ***to*** 3 **do**
        $V[\Upsilon[i]]$ = $X[i]$
        if $V[\Upsilon[i]] \neq s[\Upsilon[i]]$ **then**
            $u = u + 1$
        end
    end
    if $u \neq 0$ **then**
        p = losowa liczba rzeczywista z przedziału $(0, 1)$
        if q^u > p **then**
            Dodaj rekord V do NDB
        end
    end
    Dodaj rekord V do NDB
end
Zwróć powstałą NDB

3. Solwery SAT

3.1. Problem spełnialności

Problem spełnialności (ang. *Boolean satisfiability problem*, w skrócie SAT) polega na sprawdzeniu, czy dana formuła logiczna posiada odpowiednie przypisanie zmiennych przy których cała formuła będzie prawdą.

Wyrażenia wykorzystywane w problemie SAT składają się z:

– zmiennych - $x_1, x_2 \dots x_n$, mogących przybrać wartość prawdziwą ($true$, T, $\top$, 1) lub fałszywą ($false$, F, $\perp$, 0)

– operatora koniunkcji - AND, $\wedge$

– operatora alternatywy - OR, $\vee$

– operatora negacji - NOT, $\neg$

– nawiasów - ()

Formuły SAT przedstawia się w postaci CNF (koniunkcyjna postać normalna, ang. *conjunctive normal form*) której struktura jest następująca: formuła jest koniunkcją klauzul, a klauzula jest alternatywą literałów (tj. zmiennych lub ich negacji).

$$(x_{11} \vee x_{12} \vee \dots \vee x_{1n}) \wedge (x_{21} \vee x_{22} \vee \dots \vee x_{2n}) \wedge \dots \wedge (x_{m1} \vee x_{m2} \vee \dots \vee x_{mn})$$

Naiwny algorytm SAT jest trywialny - wystarczy dla każdej możliwej kombinacji zmiennych $x_1 \dots x_2$ sprawdzić czy formuła jest spełnialna. Sprawdzenie następuje w czasie liniowym od wartości n, zatem cały algorytm ma złożoność obliczeniową $O(2^n)$ - niepraktyczną w praktycznie każdych zastosowaniach poza bardzo małymi wartościami n.

Problem SAT jest pierwszym problemem dla którego udowodniono że jest NP-zupełny (Twierdzenie Cooka-Levina [9, 10]). Oznacza to, że można w czasie wielomianowych sprowadzić każdy problem decyzyjny zawierający się w NP do SAT.

Problem należący do NP charakteryzuje się tym, że zweryfikowanie pojedynczego rozwiązania jest możliwe w czasie wielomianowym dla deterministycznej Maszyny Turinga, oraz że sprawdzenie wszystkich możliwości (a więc rozstrzygnięcie problemu) jest możliwe również w czasie wielomianowym, ale dla niedeterministycznej Maszyny Turinga - przez jednoczesne sprawdzenie każdej kombinacji. Wynika z tego również, że $P \subseteq NP$.

Nie każda możliwa formuła CNF stanowi problem NP-zupełny. Twierdzenie o dychotomii Schaefera określa szczególne instancje problemu należące do P (przy założeniu, że $P \neq NP$) [11]:

1. Formuła jest spełnialna jeśli wszystkie zmienne przyjmują wartość 0
2. Formuła jest spełnialna jeśli wszystkie zmienne przyjmują wartość 1
3. Każda klauzula ma co najwyżej jeden literał pozytywny (każda klauzula jest klauzulą Horna)
4. Każda klauzula ma co najwyżej jeden literał negatywny (każda klauzula jest dualną klauzulą Horna)
5. Każda klauzula ma co najwyżej dwa literały (2-CNF)
6. Formuła jest równoznaczna systemowi równań linowych $x_1 \oplus x_2 \oplus \cdots \oplus x_n = c$, gdzie $x_i,\ c \in \{0, 1\}$ a $\oplus \equiv$ dodawanie mod_2

3.2. Opis działania solwerów SAT

Solwery SAT można podzielić na dwie kategorie – kompletne i niekompletne. Algorytmy kompletne gwarantują, że jeśli istnieje spełniające przypisanie dla analizowanej formuły to zostanie zwrócony wynik pozytywny. Natomiast wynikiem działania solwera niekompletnego jest albo pozytywny – istnieje odpowiednie przypisanie lub niezdefiniowany – algorytm nie znalazł rozwiązania w określonej liczbie prób.

3.2.1. Solwery kompletne

Algorytmy kompletne do rozstrzygania problemów SAT korzystają z szeregu metod do których należą: kwantyfikacja egzystencjalna, wnioskowanie, przeszukiwanie, wnioskowanie i przeszukiwanie [12].

3.2.1.1. Kwantyfikacja egzystencjalna

3.2.1.2. Wnioskowanie

3.2.1.3. Przeszukiwanie

3.2.1.4. Wnioskowanie i przeszukiwanie

3.2.2. Solwery niekompletne

Do metod niekompletnego rozstrzygania spełnialności należą: [13].

3.3. Wykorzystywanie solwerów SAT w celu uzyskania przeciwobrazu Negatywnej Bazy Danych

Problem znalezienia ukrytego ciągu w Negatywnej Bazie Danych jest równoważny SAT i zapis napisów nad alfabetem $\{0, 1, *\}$ można sprowadzić do CNF w czasie liniowym. Dla przykładu, poniższy zbiór rekordów można zinterpretować jako „Znajdź takie przypisanie wartościami {0, 1}, żeby dla każdego rekordu nie pokrywało się na co najmniej jednej pozycji".

11*0
001*
*111
*101

Tabela 3.1. Przykładowy zbiór rekordów NDB

Zatem pomijając symbole $*$, zamieniając wartości $\{0, 1\}$ na odpowiadające literały, negując je i łącząc operatorem alternatywy oraz wstawiając operatory koniunkcji pomiędzy powstałe klauzule otrzymujemy postać CNF:

$$(\neg x_1 \vee \neg x_2 \vee x_4) \wedge$$
$$(x_1 \vee x_2 \vee \neg x_3) \wedge$$
$$(\neg x_2 \vee \neg x_3 \vee \neg x_4) \wedge$$
$$(\neg x_2 \vee x_3 \vee \neg x_4)$$

Metoda działania solwerów SAT nie polega znalezieniu wszystkich rozwiązań tylko na sprawdzeniu czy dana formuła jest spełnialna, dlatego w sytuacji próby odwrócenia baz wielorekordowych lub wygenerowanych z użyciem algorytmów niekompletnych atakujący potrzebuje dla każdego pojedynczego rozwiązania dodać dodatkową formułę przeczącą znalezionemu rekordowi i ponownie uruchomić solwer.

W związku z faktem, że uzyskanie dostępu do nawet pojedynczego rekordu w systemach uwierzytelniania jest niedopuszczalne, testy są przeprowadzone pod kątem znalezienia pierwszego rozwiązania.

4. Testy algorytmów

4.1. Opis testowania za pomocą solwerów SAT

4.2. Testy metod weryfikujących poprawność rekordów pozytywnych

4.3. Testy algorytmów generacji NDB

5. Implementacja systemu uwierzytelniania

5.1. Reprezentacja danych

5.2. Algorytm tworzenia użytkownika

5.3. Algorytm uwierzytelniania użytkownika

5.4. Działanie aplikacji

6. Wnioski

6.1. Bezpieczeństwo przedstawionej implementacji

6.2. Możliwe rozszerzenia

Bibliografia

[1] Fernando Esponda. “Negative Representations of Information”. PhD thesis. 2005.

[2] F. Esponda, S. Forrest i P. Helman. „Enhancing Privacy through Negative Representations of Data”. W: 2004.

[3] Haixia Jia, Cristopher Moore i Doug Strain. „Generating Hard Satisfiable Formulas by Hiding Solutions Deceptively”. W: *Journal of Artificial Intelligence Research - JAIR* 28 (mar. 2005). DOI: *10.1613/jair.2039*.

[4] Dimitris Achlioptas, Haixia Jia i Cristopher Moore. „Hiding Satisfying Assignments: Two are Better than One”. W: *Proceedings of the National Conference on Artificial Intelligence* 24 (kw. 2005). DOI: *10.1613/jair.1681*.

[5] Fernando Esponda i in. „Protecting Data Privacy Through Hard-to-Reverse Negative Databases”. W: *Information Security*. Red. Sokratis K. Katsikas i in. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. ISBN: 978-3-540-38343-7.

[6] D. Zhao i in. „A fine-grained algorithm for generating hard-toreverse negative databases”. W: *2015 International Workshop on Artificial Immune Systems (AIS)*. 2015, s. 1–8. DOI: *10.1109/AISW.2015.7469244*.

[7] R. Liu, W. Luo i L. Yue. „The p-hidden algorithm: Hiding single databases more deeply”. W: *Immune Computation* 2 (sty. 2014), s. 43–55.

[8] R. Liu, W. Luo i X. Wang. „A Hybrid of the prefix algorithm and the q-hidden algorithm for generating single negative databases”. W: *2011 IEEE Symposium on Computational Intelligence in Cyber Security (CICS)*. 2011, s. 31–38. DOI: *10.1109/CICYBS.2011.5949400*.

[9] Stephen A. Cook. „The Complexity of Theorem-Proving Procedures”. W: *Proceedings of the Third Annual ACM Symposium on Theory of Computing*. STOC ’71. Shaker Heights, Ohio, USA: Association for Computing Machinery, 1971, 151–158. ISBN: 9781450374644. DOI: *10.1145/800157.805047*.

[10] B. A. Trakhtenbrot. „A Survey of Russian Approaches to Perebor (Brute-Force Searches) Algorithms”. W: *Annals of the History of Computing* 6.4 (1984), s. 384–400. DOI: *10.1109/MAHC.1984.10036*.

[11] Thomas J. Schaefer. „The Complexity of Satisfiability Problems". W: *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*. STOC '78. San Diego, California, USA: Association for Computing Machinery, 1978, 216–226. ISBN: 9781450374378. DOI: *10.1145/800133.804350*.

[12] Adnan Darwiche i Knot Pipatsrisawat. „Complete Algorithms". W: *Handbook of Satisfiability*. Red. Armin Biere i in. T. 185. Frontiers in Artificial Intelligence and Applications. IOS Press, 2009, s. 99–130. DOI: *10.3233/978-1-58603-929-5-99*.

[13] Henry A. Kautz, Ashish Sabharwal i Bart Selman. „Incomplete Algorithms". W: *Handbook of Satisfiability*. Red. Armin Biere i in. T. 185. Frontiers in Artificial Intelligence and Applications. IOS Press, 2009, s. 185–203. DOI: *10.3233/978-1-58603-929-5-185*.